Design and Implementation of Group Decision Support System using Object-Oriented Modeling Technique*

Soung Hie Kim** · Sung Sik Cho** · Sun Uk Kim*** · Hung Kook Park****

OMT를 이용한 그룹의사결정지원시스템의 설계 및 구현

김성희 · 조성식 · 김선욱 · 박홍국

〈Abstract〉

Recently, in organizations many decisions are being made by groups. And the organization is changing a lot and so are groups. To help decision making of changing groups, we need more flexible and more adaptive GDSS. Therefore one of the critical success factors of GDSS is flexibility and incremental improvement.

Prior research on specifying design requirements of GDSS suggests generic design requirements. But they are too general to be incorporated directly into system design, because of the disparity between real group and ideal group that the researchers studied. Many design strategies that start from the generic design requirements thus have contingency variables that change as the characteristics of group change. From the viewpoint of developers, these variables implicate the desirability of flexibility. To achieve flexibility we need new methodology of design and implementation.

Nowadays, object-oriented analysis and design methodologies have been progressed to the point that many systems are being developed through these methodologies.

In this paper, a design is proposed using Object-Oriented Modeling Technique (OMT). Exploiting object-oriented paradigm results in a highly flexible and easily upgradable design.

1. Introduction

A group or work group refers to two or more individuals who act as one unit to achieve a common goal. The group can be permanent or temporary. The group can be in one location or in several locations,

* This paper was supported by Korea Science and Engineering Foundation(KOSEF# 93-0100-12-01-3).

** Dept. of Management Engineering, Graduate School of Management, KAIST, Seoul, Korea

*** Dept. of Industrial Engineering, DanKook University, CheonAn, Korea

**** Dept. of Telecommunication Systems Management, SangMyung University, Seoul, Korea

and it can meet at the same time or at different time. A group can be a committee, a review panel, a task force, an executive board, a team, or a permanent unit.

A Group Decision Support System (GDSS) is an interactive, computer-based system that facilitates the solution of unstructured problems by a set of decision makers working together as a group [1]. And a GDSS consists of a set of software, hardware, language components, and procedures that support a group of people engaged in a decision-related meeting [2].

Work environment tends to change a lot and so do groups. Deep transformations are modifying today's organization of work. Changes in the social environment permeated work settings with a whole new regime of demands and constraints. The turbulence characterizing modern business environments forced industrial enterprises to improve their innovative skills, operation flexibility and product quality. To meet these demands, work organization must change rapidly and needs more flexible and more adaptive GDSS to help decision making for changing groups.

One of the critical success factors of GDSS is incremental improvement [3][4]. Using feedback of the participants with hardware and software innovations, the GDSS facility needs to be enhanced constantly. Groups in current work environment keep changing, thus a GDSS should have flexible design and architecture to adapt itself to changing environment. To make GDSS have these property, new methodology of design and implementation is needed.

On the technological side, since the mid-1980's there has been growing interest among system researchers in an alternative approach to system development called object-oriented analysis and design [5]. The advantage of this approach can be summarized as: better user/analyst communication and thus easier modeling, reusability of code, and improved flexibility. Nowadays this approach is mature to create many design and implementation methodologies. One of them is Object-Oriented Modeling Technique (OMT)[6].

OMT presents an object-oriented approach to software development based on modeling objects from the real world and then using the model to build a language-independent design organized around those objects. Object-Oriented models are useful to understand problems and design programs and databases. A design using OMT tends to have much flexibility and maintainability.

The main concern of this research is to design and its implement a GDSS appropriate for changing work environment using OMT. By using object-oriented paradigm, the design proposed is flexible, maintainable and can be evolved easily.

The scope of this research is the design and implementation of a GDSS for meetings that generates new ideas. There are various meetings in real work environment such as reporting, decision making, generating new ideas and project [7]. The basic functionality of GDSS for meetings of generating new ideas include idea generation, idea organization, voting, and documentation.

The rest of this paper is organized as follows. Section 2 presents the multi-disciplinary research on the groups and the design strategies, and discusses the limitations of current GDSS design. Section 3 reviews OMT briefly and explains why OMT is chosen as a development methodology. Section 4 proposes a design of GDSS. This design is based on the three-layered Client/Server architecture. Section 5 explains how our design is implemented. The final section summarizes the contributions of this paper and discusses a direction of further studies.

2. Literature Review

2.1 GDSS Foundation Issues

This section describes a survey and analysis of GDSS research foundations that are used to guide GDSS

〈Table 2.1〉 Work Group Research(Source: Mandviwalla, 1994)

Author	Field	Focus
Alderfer (1987)	industrial psychology	inter-group issues
Friedlander (1987)	organizational development/behavior	group as a subsystem of the organization
Gersik (1989)	organizational behavior	group development
Goodman, Ravlin, & Schmike (1990)	organizational behavior	effectiveness and underlying assumptions
Hackman (1987)	organizational behavior	design of and process gains
Hirokawa & Johnston (1989)	communications	communication, development & decision making
Levine & Moreland (1988)	social psychology	group processes
Mintzberg (1973)	management science	use of time
McGrath (1984)	social psychology	group tasks
McGrath (1991)	social psychology	task, time, & development
Panko (1992)	information systems	use of time & characteristics
Schwartzman (1990)	organizational behavior	meetings
Steiner (1972)	social psychology	process losses & development
Stokols & Shumaker (1981)	organizational behavior	settings & environment
Sundstrom, De Meuse, & Futrell (1990)	industrial psychology	effectiveness & types
Wells (1990)	sociology	intact groups
Wicker (1987)	environmental psychology	environment

development. Most modern system development methodologies emphasize the importance of focusing on user needs. Therefore, the starting point of these methodologies is the definition of user requirements that must be reflected into the system design. GDSS development is no exception.

〈Table 2.1〉 describes a multi-disciplinary survey and analysis of work group research that is used to derive generic GDSS design requirements. The work group is studied in many disciplinary areas, including organizational behavior, organizational dynamics, management science, social psychology, decision theory, communications, and office automation. From the viewpoint of GDSS design, all these disciplines are valid reference points providing clues to identifying GDSS design requirements. This paper limits the definition of work group as two or more individuals whose mission is to perform some tasks and who act as one unit. The group can be permanent or temporary [8].

Based on the prior research many design requirements were derived as shown in 〈Table 2.2〉 however these requirements are too general for developers to reflect them into system design. Thus, these design requirements remain as an abstraction of contingency variables such as group size, proximity, and tasks.

〈Table 2.2〉 Generic Design Requirements [9]

Generic Design Requirements
Support multiple group tasks
Support multiple work methods
Support the development of group
Provide interchangeable interaction methods
Sustain multiple behavioral characteristics
Accommodate permeable group boundaries
Adjustable to the group's context

〈Table 2.3〉 summarizes strategies for designing GDSS. DeSanctis and Gallupe proposed a multidimensional taxonomy of GDSSs using the contingency approach. They suggested that the design of GDSS should be driven by three factors: the size of the group, the presence or absence of face-to-face interaction, and the task confronting the group [10].

Huber suggested an activity-driven design strategy as an alternative to the task-driven strategy [2]. Decision groups can have a wide variety of tasks. To hold various tasks in a general GDSS, he suggested that design should focus on group activities rather than on group tasks. And he insisted that whatever task a group may engage in, group members will be found to be carrying out one or more of the following activities: information retrieval or generation, and information sharing or information use.

Nunamaker et al. focused on the process gains and losses [11]. They proposed four contingency variables (group, task, context, GSS (Group Support System)) that affect the meeting process, meeting outcomes and four theoretical mechanisms by which the GSS can affect the balance of gains and losses: process support, process structure, task support, and task structure. And they proposed three basic concepts on which the general design builds: GSS meeting room, meeting room facilitation, and software toolkit.

〈Table 2.3〉 GDSS Foundation Issues

Author	Focus	Design Strategy
DeSanctis & Gallupe	Information-Exchange of Decision Making	Group Size, Member Proximity, Task
Huber	Information Sharing & Use	Activity-Driven Design
Nunamaker, et al.	Process Gains & Losses	Process Support, Task Support, Process Structure, Task Structure

2.2 Limitations of Current GDSS Design

This section outlines the limitations of the extant GDSS. However, the limitations do not necessarily indicate that existing GDSSs are poor products. Instead the limitations outlined here indicates the need for another design approach that can make up the design limitations.

Cannot support multiple group tasks

One way to provide support for multiple group tasks is to implement systems patterned into different modules that support each generalized task process. For example, *VisionQuest* supports multiple group tasks through a toolkit of modules that directly map into generalized group tasks such as generating ideas(e.g. brainstorming tool). However, the depth of support for a particular task is limited. For example, the feature set is limited in systems such as *VisionQuest* compared with specialized tools such as *ForComment* for managing documents related to group meetings and conferences and *TALKShow* for desktop conferencing. However, these systems in turn are limited in that they are often self contained and focused on addressing one particular task. Developing practical support for truly multiple group tasks will entail massively expanding the functionality of a single product or getting products from different vendors to work together.

In contrast, other systems such as *Lotus Notes* ignore support for specific tasks and are oriented to information sharing and communication. Lotus Notes provides only general foundation and leaves generating task specific modules to users or third-party vendors.

Designed to fit in special settings

GroupSystems [12] was designed for decision room that consists of a large, usually U-shaped table equipped with twelve to thirty networked microcomputers and a large screen projection system that permits the display of work done at individual workstations. Although most of the offices have special rooms for meetings or discussions, they are usually not equipped as decision rooms but with just a table and an OHP. The systems that need additional equipment fail to permeate into real life, thus remaining in the laboratory. It does not seem reasonable to expect a group to use separate systems for brainstorming in face-to-face and distributed meetings, while receiving an e-mail and documents related to the brainstorming topic on another system.

Designed on simplified view of groups

A few theories and systems take an "egalitarian" approach toward group decision making, focusing only on the positive aspects of collaboration while ignoring the status, power, and interest differences among group members [13][14]. Other systems have been designed to eliminate or reduce "process losses" and direct group development on to a theoretically ideal stage-based sequential path. Process losses is the term coined by Steiner [15] to describe deficiencies in group performance due to actions by group members that contribute negatively or do not contribute positively to group performance. Actions that can cause process losses include socialization and conformance to pressure. Hackman [16] questions the usefulness of the "process losses" view, arguing that it limits the ability to visualize process gain. McGrath [17] suggests that the group process is too complicated to be simply labeled as positive or negative and views some "process losses" as necessary part of group interaction.

Gersik [18][19] concluded from a series of experiments that work groups do not follow phase-based or stage-based paths of development described by earlier models such as Tuckman's [20] forming, storming, norming, and performing. The systems developed on these theories cannot survive because of the disparity between real groups and ideal groups. These systems can meet only a special instance of groups.

Most modern system development methodologies emphasize the importance of focusing on user needs. All of us have experience with groups and we automatically construct an "average" group that reflects our intuitions and experiences. Yet, there is no "average" group because every group is a unique combination of its constituents, environment, and task. It is hard to predict group requirements; groups are dynamic entities that change with time especially in real business environment. Finally, the principles of developing applications for groups are not necessarily the same as those of developing applications for individuals.

2.3 Summary and Discussion

Prior researches have been done to define the design requirements of GDSS. GDSS developers are given several design guidelines or design requirements that they must obey. It is hard for them to implement all the design requirements suggested by the prior researchers. And it is meaningless to develop such systems because the real group who will use the systems is not the same group in the design framework. Furthermore, no matter how well a GDSS system is designed, if the group or organization that uses the GDSS changes, it is likely that it will fail or be used sub-optimally.

In spite of a lot of research on groups shown in 〈Table 2.1〉, the design requirements of GDSS have not been specified clearly yet, because the group is not fixed. Group is a dynamically moving target. Therefore, the design strategies proposed by prior researchers in 〈Table 2.3〉 had to use contingency variables or had to be general. From the perspective of GDSS developers, they should reflect these variables in the design. But if

one of the contingency variable, for example, task, changes the developer should rebuild the system. For this reason, flexibility and incremental improvement of system became the essential critical success factors of GDSS [3][4].

In summary, many researchers studied groups and the output has been used to derive the design requirements of GDSS. But a group is not a fixed target. The design requirements are contingent to the situation the group is in. Therefore, the design strategies have to emphasize the flexibility and incremental improvement to meet the contingent requirements. It means that a key factor for GDSS success is the development methodology that can overcome the limitations of the previous development methodologies.

3. OMT Approach

3.1 An Overview of OMT

OMT is chosen as the design method of GDSS. This section reviews what OMT is and how it works.

3.1.1 Strategy

According to Rumbaugh, the overall OMT strategy of system development is as follows [6][21]:

Conceptualization: Conceive a problem to be solved and a system approach that solves it. Make an initial cut at the problem statement by writing use cases or listing requirements.

Analysis: Starting from a statement of the problem, the analyst builds a model of the real-world situation showing its important properties. The analysis model is a concise, precise abstraction of *what* the desired system must do, not *how* it will be done. The final analysis models are the true requirements (although they may continue to change as requirements evolve or mistakes are discovered).

System Design: During system design, the target system is organized into subsystems based on both the analysis structure and the proposed architecture.

Object Design: The object designer builds a design model based on the analysis model but containing implementation details. The focus of object design is on the data structure and algorithms needed to implement each class.

Implementation: The object classes and relationships developed during object design are finally translated into a particular programming language, database, or hardware implementation.

There is no sharp line between *conceptualization* and *analysis*. Rumbaugh used the word *conceptualization* for the initial informal attempt to express the system goals in words, and analysis for a more rigorous attempt to build and understand models to capture the requirements.

3.1.2 Three Models

The OMT uses three modellings to describe a system:

Object modeling: It describes the static structure of the objects in a system and their relationships. The object model contains object diagrams. Object diagram is a graph whose nodes are object classes and whose arcs are relationships among classes.

Dynamic modeling: It describes the aspects of a system that change over time. The dynamic model is used to specify and implement the control aspects of a system. The dynamic model contains state diagrams. A state diagram is a graph whose nodes are states and whose arcs are transitions between states caused by events.

Functional modeling: It describes the data value transformations within a system. The functional model contains data flow diagram.

The three models are orthogonal parts of the description of a complete system and are cross-linked. The object model is fundamental, however, because it is necessary to describe *what* is changing or transforming before describing *when* or *how* it changes.

3.2 Why OMT?

Object-oriented development inverts the previous function-oriented methodology, as exemplified by the methodologies of Yourdon [22] and DeMarco [23]. In these methodologies, primary emphasis is placed on specifying and decomposing system functionality. By contrast, the object-oriented approach focuses first on identifying objects from the application domain, then fitting procedures around them. Object-oriented software holds up better as requirements evolve, because it is based on the underlying framework of the application domain itself, rather than the ad-hoc functional requirements of a single problem.

A comparison of recent object-oriented methods [24] bases the comparison on features such as concepts, models, and support of the process concept, but not on the actual experience with the methodologies. From the methods discussed, OMT is selected because of its strong support for analysis and design as well as implementation. Wirfs-Brock's methodology, known as CRC (Class, Responsibility, Collaboration) [25], is simplistic but very practical method. Much of its notation is poor and it does not cover all the issues. It is usually used as a precursor to the use of other notations. It is good for the requirement capture stage and as a pedagogical tool. Buhr's methodology [26][27], is tied to Ada packages. HOOD (Hierarchical Object-Oriented Design) [5] still relies on SA/SD for the requirements analysis. The lack of support for inheritance makes it object-based rather than object-oriented. Thus, reuse is supported but not extensibility. Booch's methodology [28] looked more powerful than OMT, but OMT was chosen because it seemed to strike the right balance between simplicity and expressive power.

3.3 Availability of OMT in GDSS Domain

3.3.1 Flexibility

GDSS design frameworks have many contingent variables (e.g. group size, task) that implicate flexibility to the developers.

Flexibility has various definitions in various domains. For example, in a CSCW (Computer Supported Cooperative Work) system, flexibility is the ability to switch dynamically between different states or modes of a CSCW tool. It means that a tool is more flexible if it supports various applications rather than only one application.

In GDSS domain, flexibility has a little different meaning. GDSS should provide a set of software tools, similar to a DSS model base, that is a collection of generic tools for various group activities such as idea generation and voting rather than being one indivisible system to support the entire task, for example, strategic planning. Each tool provides a different approach to support a particular activity; thus the GDSS can provide various combinations and styles of process structure, process support, task structure, and task support during any one meeting. Groups use many different approaches and often do not proceed in a straightforward manner. The tools can easily be mixed and combined with non-GDSS activities in whatever order the group believes is most effective. This philosophy also enables new tools to be easily added to the toolkit and existing tools to be customized to specific needs.

By inheritance property, OMT can support building a set of tools. Whenever each tool is built, it is needless to build from the bottom. More specific objects can be derived from the general objects by inheritance mechanism. It is in other words reusability. From the basic design requirements a set of basic tools is built and then both the design and the code can be reused to make more specific tools that support various tasks.

OMT helps rapid system evolution because of its

better communication between user and analyst. When the user requirements or the situation change OMT can support making new tools rapidly. GDSS design framework cannot cover all the situations before system development. The limitations of design can be made up with the incremental improvement by using OMT.

3.3.2 Coordination

Although flexibility is important, it is also important to restrict the number and type of functions available to participants [29]. Restrictiveness provides a more powerful intervention so that groups are more likely to use the GDSS as intended by its designers, which has been one of the problems with non-computerized techniques [30]. Restrictiveness promotes the use of more effective techniques and prevents less effective ones, fosters learning, promotes consistency, and provides coordination to ensure that all group members are using the same tool at the same time or consistently. In this context the GroupSystems is locally restrictive so that users can perform only certain functions while providing flexibility in that a wide variety of tools are available.

To achieve this restrictiveness, coordination among the tools is needed. By using OMT, we can consider all the tools as objects that communicate each other. Object-oriented concept helps building coordination mechanism by its message passing mechanism. A special object in the lower layer that controls the communication among objects on the higher layer is needed. But the coordination mechanism can be changed when a new object is added into the higher layer or when another mechanism is needed. Therefore, it is desirable that the change of coordination mechanism must not affect the communication of other objects. The special object can encapsulate the coordination mechanism without affecting other objects.

4. Design

4.1 Problem Statement

In this section design starts from the problem statement. This problem statement includes basic functionality that GDSSs should have [31]. Some additional requirements from real work groups.

4.1.1 Basic Functionality

Idea generation meetings need following basic functionality:

- *Agenda*: A description of what activities have to be performed and when.
- *Idea Generation*: An activity that allows participants to share ideas simultaneously and anonymously on a specific topic proposed to the group. Participants are able to contribute ideas at the same time with no loss of information. After a participant enters an idea, it can be seen to other participants on the public screen. Idea generation tool gathers ideas and saves it for future use.
- *Idea Organization*: An activity that assists groups categorizing the ideas generated in idea generation.
- *Voting*: An activity that offers a method for polling opinions of the group members. It prioritizes alternatives and determines the degree of group consensus.
- *Documentation*: An activity that makes report for participants. In the report all information about the meeting is described and it can be used for future meetings.

4.1.2 Additional Requirements

From the current work environment some factors are found:

Meeting on the desk: In current work environment, most of the work is accomplished on the desk. The meetings are no exception. Now, work environment is

based on computer-network system, therefore it is an additional burden to make and manage special room (decision room) for meeting. Whole the office is a decision room.

Meeting without facilitator: Of course it is better to have an expert(facilitator) for meetings. But usually in the current work environment, there is not any facilitator. So GDSS should be able to support the group who does not have a facilitator.

Freedom to hold a meeting: Anyone who needs a meeting should be able to schedule a meeting, be able to be a meeting leader and be able to choose participants who are necessary to the meeting. People who have problem to discuss or to decide know the context of the problem the best. So, it is natural for people who need a meeting to lead the meeting.

Easy access to the document of meetings: It means providing database access for previous meetings. If people need information about some problem then he/she could look up the previous meeting documents. It does not matter whether he/she was the participant of the meeting or not.

Idea generation anytime before voting: This feature enables the asynchronous meeting. For some reason participant cannot participate the meeting, but he/she can adds his/her ideas to the meeting when he/she feels free. This feature can help get rid of time barrier of the meeting.

4.1.3 Problem Statement

A part of problem statement is shown in 〈Figure 4.1〉. A problem statement is a statement of requirements. This problem statement reflects the basic functionality and the additional requirements shown in section 4.1.1 and 4.1.2. This problem statement is not complete. There can be more problem statement that describes each activity.

〈Figure 4.1〉 Problem Statement of GDSS (*part*)

Meeting members are one leader and one or more participants.
If someone needs a meeting, then he/she creates new meeting and he/she becomes the leader of the meeting.
The leader enters title of the meeting and description of the meeting.
The description includes why the leader wants the meeting briefly.
The leader then browses the users of the GDSS and selects the participant who are needed for the meeting.
The leader decides in what procedure the meeting will proceed. Usually the meeting procedure is brainstorming, idea organization, voting, reporting.
After the leader completed creating new meeting, he/she informs the participants of the new meeting.
When the meeting begins participants executes the GDSS and logins to the server.
The Leader can browse who are logged in.
The participants open the meeting that the leader created, and read the meeting description and meeting procedure.
The participants execute activities such as brainstorming, voting.
The participants can join any activity if the activity is in accord with the whole procedure.
The participants can join brainstorming at any time before the voting begins.
After the brainstorming the leader categorizes the ideas.
The meeting can be adjourned if the leader allows.
After the meeting is over, everyone who has an account of the GDSS can browse the result of the meeting.
Everyone can make report of previous meeting by using reporting tool.
The meeting can be proceed anonymously or not.
A participants can communicate with other participants informally during the meeting.

4.2 Analysis

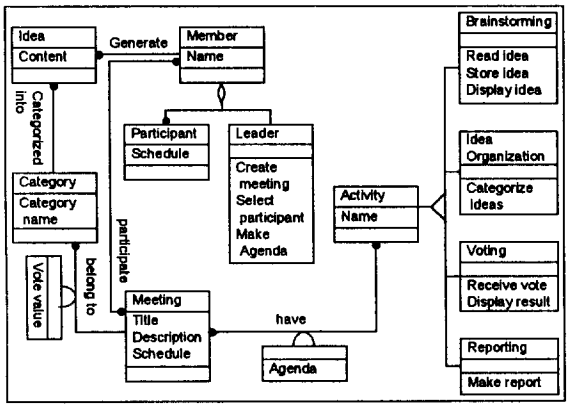
From the problem statement, three models are derived; object model, dynamic model, and functional model.

4.2.1 Object Modeling

The object model is very simple. The details are expressed in the design step. This model does not include implementation-related objects like communication objects. These objects are added in the design step.

The first step in constructing an object model is to identify relevant object classes from the problem

statement. Not all classes are explicit in the problem statement; some are implicit in the application domain or general knowledge. Classes often correspond to nouns. The second step is to discard unnecessary and incorrect classes such as redundant classes, irrelevant classes, and implementation constructs. And the third step is to identify the associations and inheritance among the classes.

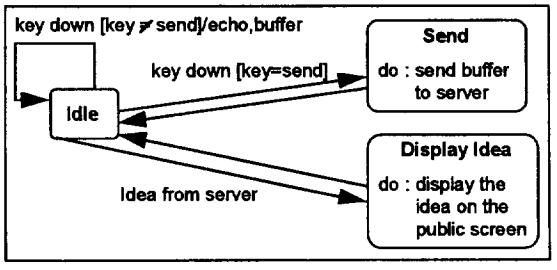


〈Figure 4.2〉 Object Model of Meeting

4.2.2 Dynamic Modeling

An object model describes the possible patterns of objects, attributes, and links that can exist in a system. The attribute values and links held by an object are called its state. Over time, the objects stimulate each other, resulting in a series of changes to their states. An individual stimulus from one object to another is an event. The response to an event depends on the state of the object receiving it, and can include a change of state or the sending of another event to the original sender or to a third object. The pattern of events, states, and state transitions for a given class can be abstracted and represented as a state diagram.

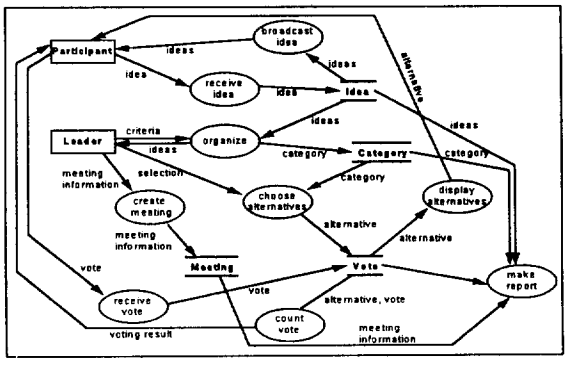
Each object has its dynamic model, that describes control aspects of the object. The objects with no operation do not have dynamic models, because their states do not change. In this section, the dynamic model of *Brainstorming* object is described for example. Brainstorming object sits idle until a participant enters an idea or an idea arrives from other participant. When it receives an idea from a participant, it moves to send state and does the sending activity then it goes back to idle state. When it receives an idea from other participant, it moves to display state and displays the idea and then goes back to idle state. Brainstorming moves among these states. The dynamic modeling describes the mechanism that reacts to the external stimuli.



〈Figure 4.3〉 Dynamic Model of Brainstorming Object

4.2.3 Functional Modeling

〈Figure 4.4〉 shows the functional model of a GDSS. It is the same as the data flow diagram. Functional model shows which values depend on which other values and the functions that relate them.



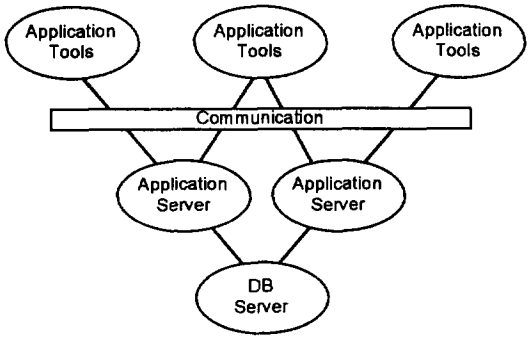
〈Figure 4.4〉 Functional Model of Meeting

The rectangles are actors, the ellipses are processes, the arcs are flows of data, and a pair of thick lines mean a data store. The actors are external to a meeting and others are internal to a meeting. Thus, the functional model analyzes the data that are communicated between the users and the system. In this functional model the actors are only a leader and participants. The data stores corresponds to the objects that do not have operations in the object model shown in 〈Figure 4.2〉. The processes are the operations in the objects.

4.3 Design

4.3.1 System Architecture

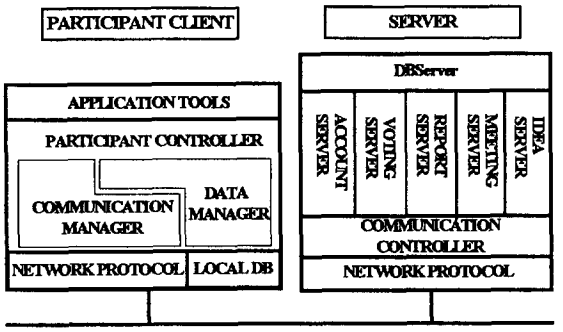
In this section three-layered Client/Server architecture is proposed. Three-layered Client/Server architecture allows more flexibility in handling domain-specific coordination needs. 〈Figure 4.5〉 shows the abstract architecture.



〈Figure 4.5〉 Abstract System Architecture

- DB Server : Provides basic database access functions.
- Application Servers : Take care of coordination and other needs associated with the execution of a particular application.
- Application Tools : Provide user interface and application tools.

More details are following;



〈Figure 4.6〉 Detailed System Architecture

In this detailed system architecture, DBMS and Communication Controller are included in GDSS Server, the first layer. And all the servers that are placed between DBMS and Communication Controller are Application Servers, the second layer. And all the components in Participant Client is included in the third layer, Client layer.

Each component of 〈Figure 4.6〉 is described as follows;

1. DB Server :
   - *DBMS* : All the data service is provided here. It processes the data requests from the application layers.
2. Application Server :
   - *Idea Server* : It receives all the ideas from the clients, asks DBMS to store them and requests Communication Controller to send out them to other clients, so that all the participants can see the ideas.
   - *Meeting Server* : It manages all the information about the meeting environment. It manages which members participate in which meeting, which agenda the meeting follows, and what activity the meeting is in. There can be several meetings simultaneously, in such a case meeting server

manages all the meetings.

· *Report Server* : It replies to the requests of documenting the meeting results. It gathers the information about a certain meeting, makes report and then sends to the client who made the request. Participants can request the report of previous meetings to prepare a new meeting.

· *Account Server* : It manages the accounts of the users. It must recognize who are logged in and their personal information.

3. Application Tools :

· *Application Tools* : These tools are the end point that users interact. They are *Brainstorming tool, Voting tool, Reporting tool, Participants selection tool.* They are actually the menu items of client system.

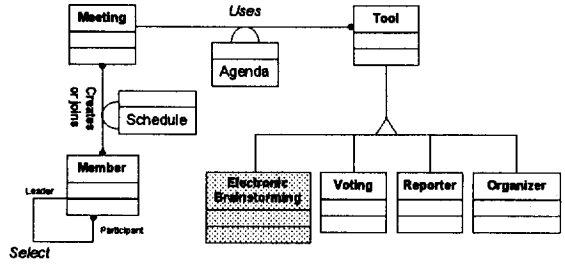
4. Controller :

· *Communication Controller* : It routes messages from the clients to application servers. It decides which message has to be passed to which object. Clients send various messages, for example, ideas, voting information, and then routes them to appropriate application servers such as *Idea Server* or *Voting Server.* And it also sends out the messages from the application layer to the clients.

· *Participant Controller* : It provides basic user environment to the participants. It processes communication requests and data requests. It receives the ideas or voting information from the users and sends them out to the server. Also it receives the messages from the server and routes it to the application tools, such as *Brainstorming*, *Voting*. And it receives the data access requests, then it decides where the data is located and if the data is on the server it sends the requests to the server.

4.3.2 Object Design

In this step, the object model derived from the analysis step is revised. Some system-dependent objects such as database are added, and unnecessary objects are removed. 〈Figure 4.7〉 shows the object design of the overall system.

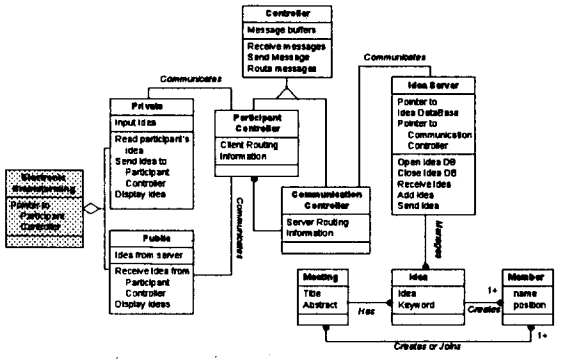


〈Figure 4.7〉 Object Design of GDSS

A person can be a leader or a simple participant. A leader creates a meeting and participants join the meeting. *Agenda* and *Schedule* are link attributes. Instead of *Activity* object in the analysis step, *Tool* object is used. It's because an executable system is being designed in this step, an activity is considered as a tool or a utility that users utilize. *Tool* object will be a menu item of the implemented system. The objects *Member*, *Schedule*, *Meeting* and *Agenda* will be a database. These objects contain data rather than operations. But the objects that belong to *Tool* object - *Electronic Brainstorming*, *Voting*, *Search*, *Categorizer* - contain operations rather than data. These objects use network and database objects. In 〈Figure 4.7〉, network object and database object are omitted. These objects will discussed on the next section.

〈Figure 4.8〉 shows the *Electronic Brainstorming* object in detail. *Electronic Brainstorming* object is composed of two objects *Private* and *Public*. These objects correspond to private screen and public screen respectively. Through *Private* object participants enter their ideas. And *Public* object shows the ideas that all participants entered. Both of these two objects are connected to *Idea Server* object. *Idea Server* object manages the *idea* object. In this design *Private* and

Public do not have to know how to manage the database. They just have to know where the *Idea Server* is. Three objects *Member*, *Idea*, *Meeting* are actually databases. If these objects are implemented in a certain vendor's database, only the *Idea Server* object must be changed, not influencing other objects. That is one of the advantages that object-oriented concepts have.



〈Figure 4.8〉 Object Design of Electronic Brainstorming.

In this object design the objects *Idea Server*, *Member*, *Idea*, *Meeting* are going to be located on the server, and other objects are going to be located on the client.

4.4 Coordination

Application layer hides internal database from the client, which prevents clients from accessing internal database illegally, thus the data consistency is checked first in the application layer before the DBMS checks. Each request from the client must go through the application layer, and each component of application layer asks *Meeting Server* to validate the request. *Meeting Server* looks up the *Agenda* object and validates the request whether it is legal or not. *Meeting Server* allows some flexibility by providing coordination rule in addition to *Agenda*. This coordination rule also controls the accessibility of participants to the document of secrete meetings. The rule can prevent some people from accessing the prohibited data by referring the *Account Server*. *Account Server* checks if people have the right to access to the secrete meetings. *Meeting Server* sets the authority of participants when it creates a meeting.

In real life some participants can not join the meeting because they are doing another work with others, in this case they need the GDSS that support different-time/same-place or different-time/different-place. To support these property participants should be able to generate ideas until the meeting is closed with voting. It means some activities can be done at different time. For example, a participant can enter his/her ideas after other participants finished brainstorming. He/she can enter his/her ideas while browsing the ideas entered before. But this can not be allowed after the voting is executed. *Meeting Server* contains this coordination rule.

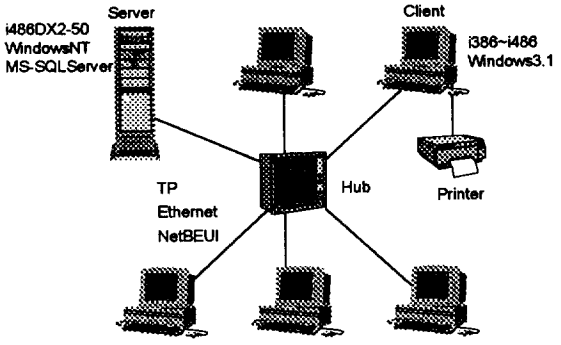
- Brainstorming is always possible before voting.
- Reporting is always possible.
- Categorization is possible after brainstorming starts and before voting.
- Else, follow the Agenda.

5. Implementation Issues

5.1 System Specification

The design is implemented on the following system specification. The system configurations of work environment have been moving from the centralized one to Client/Server or distributed one. Usually the Client/Server system has been composed of UNIX host and DOS or Windows based PCs. Recently, the PC's ability as a server has enhanced without increase of cost. Now, many work offices use PCs for the server and client. Also, there are powerful operating systems that can accomplish server's roles on the PCs, for example Windows NT or NetWare.

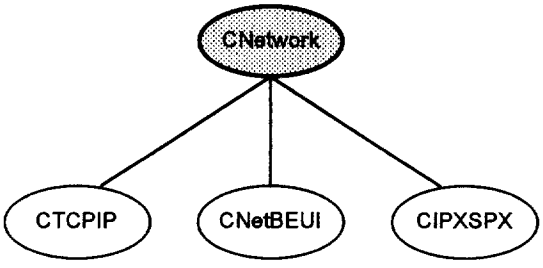
As the UNIX machines are replaced by PCs, the



〈Figure 5.1〉 System configuration of GDSS

network protocol changes to be appropriate for the new environment. TCP/IP is prevalent but it's not appropriate for small network because it was designed for large networks such as Internet. NetBEUI is designed for rather small network, so it is efficient in the office.

The flexibility or scalability have to be considered. By the object-oriented paradigm the network module can be easily substituted by another module that follows different network protocol. Indeed the network module is implemented with a protocol-independent object. The network object, *CNetwork*, is a meta object. *CNetwork* first identifies the type of network protocol and then loads appropriate DLL (Dynamic Linkage Library) that actually accomplishes the communication. Fortunately there are many network DLLs in public domain. Only what has to be done is to make meta object and to load the appropriate library.



〈Figure 5.2〉 Class Hierarchy of Network Objects

There is no need to use only object-oriented programming languages but C++ is used because it is easy to code objects. A system implemented in C++ is easier to be reused by other developers and reversely it can absorb other systems into itself, because C++ provides powerful linkage ability. And C++ can be ported to any platform, which is a essential requirement of system development.

Here's a tip of code that implement *IdeaServer* object.

```
class CIdeaServer : public Cobject
{
public:
CIdeaServer( ); //Constructor
~CIdeaServer( ); //Destructor
// attributes
LOGINREC* m_login;
DBPROCESS* m_dbproc;
CCommCntrller* m_pCommCtrller;
CIdea m_IdeaBuffer;
// methods
virtual void InitDB(DBPROCESS*, LOGINREC*);
virtual void CloseDB(DBPROCESS*, LOGINREC*);
virtual void AddIdea(DBPROCESS*, LOGINREC*, CIdea*);
virtual void SendIdea(CIdea*, CCommCntrller*);
afx void OnRcvIdea(CIdea*)
private:
virtual void MakeMsg(CMsg*, CIdea);
virtual void ResolveMsg(CMsg*);
.....
```

Some of the objects derived during the Object Design step, such as *idea*, are actually database objects. These objects are implemented with RDBMS, MS-SQLServer. MS-SQLServer provides *DB-Library* that interfaces with C++. Through *DB-Library*, C++ can use SQL.

5.2 Operation Model (A Case)

In this section, the operation of this system is explained as the meeting proceeds. From the creation of new meeting to reporting, how the system components work together is explained.

Login

Every user who wants to use this system must log in to the server. The user enters his/her login ID, password, and server's name. This information is sent to the server and *Communication Controller* in the server routes it to *Account Server*, then *Account Server* looks up the Member table and verifies that the user account is valid or not. If the user's account is valid *Account Server* updates the Member table that the user logged in.

Creating A New Meeting

User clicks the menu item 'New', then a dialog box pops up. The user enters the title, and short description of the new meeting. This information is sent to *Meeting Server* in the server, and then *Meeting Server* adds an identifier of the meeting and stores it in the Meeting table. And the *Meeting Server* considers the user who created the new meeting as the leader of the meeting.

Selecting Participants

When the leader finished creating the new meeting he/she selects the participants who are necessary for the meeting. Participant dialog box pops up when the leader clicks the menu item 'Participants'. *Account Server* reads the member table and sends the user information to the leader who clicked the 'Participants' menu. The leader selects among the members and the UserIDs of selected members are sent to the *Meeting Server* and it stores them in Participation table.

Creating Agenda

The leader now creates agenda by selecting the activities. The leader selects an activity and enters time duration now long he/she will use the activity. Meeting Server receives this information and stores it in Agenda table.

After the leader completes up to this step it's ready to start a meeting. The members who are selected by the leader don't know they were designated for the participants. Meeting Server informs them of the new meeting.

Joining a Meeting

Now a participant logs in to the server to participate in the meeting that needs him/her. *Meeting Server* shows him/her all the meetings. He/she can browse all the meetings and selects one of which he/she was informed. From now on all activities he/she does is onto the meeting he/she opened until he/she closes it.

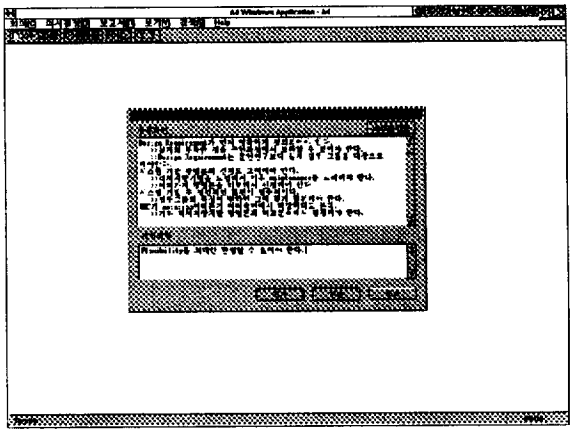
Brainstorming

A participant joins brainstorming activity. He/she enters his/her idea in the private area, and then this idea is sent to the *Idea Server* and *Idea Server* stores it in the Idea table, looks up the Participation table to find out who are participating the same meeting and then sends out the idea to the participant who are participating in the same meeting. Now participants can see the idea in the public area of their windows. A participant can add his/her idea to other's idea by double clicking the idea in the public area. The added idea is displayed under the target idea, and these ideas are categorized into the same category. Of course participants can move their ideas to anywhere only if the ideas are their own.

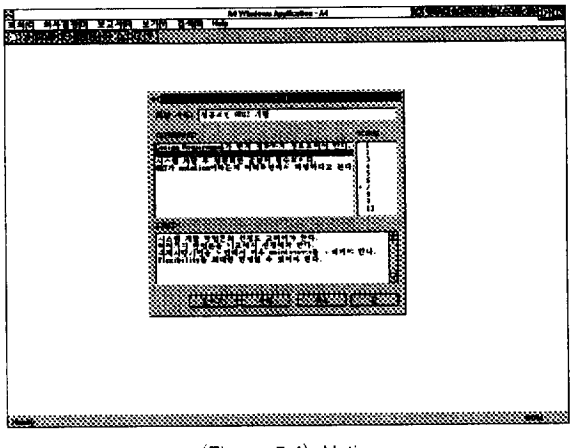
Voting

During the brainstorming many ideas were generated. Some of these ideas have been categorized already by the participants and the rest of ideas that are not categorized can be categorized by the leader. If the ideas are categorized, the categories can be the alternatives for voting. Participants give vote to each alternatives. There can be many voting methods. 5-point scale or 10-point scale is usual. When the participants finishes voting this information is sent to the *Voting Server*. It gathers the votes from the participants and stores it in

Category table. It counts the votes and shows the result to the participants.



〈Figure 5.3〉 Brainstorming



〈Figure 5.4〉 Voting

Reporting

After a meeting is finished a member can request a report. Report Server reads all the tables related to the meeting and extracts the information. Report Server sends this information to the member's computer, then the member can just browse it or save it on his/her local area. The report includes who made the meeting, the participants, title and description of the meeting, what ideas were generated, and what the result was.

Closing a Meeting

Everyone can leave the meeting by closing the meeting or exiting their client system. But other people can still continue the meeting unless all the participants log out. When the user exits the system all the meetings that he/she opened are closed.

6. CONCLUSIONS

6.1 Summary of Contributions

This research is an effort to propose a design and implementation of GDSS using OMT. By using OMT, the object-oriented concepts could be applied in GDSS domain. Previous GDSS systems have some limitations because they have trouble catching up with the work groups, the target of the systems, that change constantly. In comparison with previous GDSS systems this study designed and implemented GDSS emphasizing flexibility that is one of the critical success factors of GDSS. From the basic functionality that GDSS should support, this study described problem statement and designed basic GDSS. Also this study added additional requirements that are found in real work environment. And by object-oriented concepts, the brainstorming objects could be easily upgraded with modifying small part of the system.

This paper proposed three-layered Client/Server architecture for GDSS for flexibility and coordination. Three-layered architecture separates application layer from the server layer, thus it can support domain-specific coordination flexibility. The message passing mechanism of objects also helped implementing three layers.

GDSS developers may employ our design and implementation to their development. Because our design and implementation are object-oriented they can employ them to develop object-oriented applications with minimal effort.

6.2 Suggestions for Further Research

First, it is needed to extend this design to include various kinds of meetings such as planning, reporting, and negotiation. Because our problem statement included only the requirements of idea generation meetings, our design cannot support other kinds of meetings. In order to survive in work environment our design should support various meetings that happen everyday in business organizations.

Second, it is needed to extend our design to take advantage of new technologies. Nowadays many technologies are evolving especially in the CSCW (Computer Supported Cooperative Work) domain such as videoconferencing. It is needed to enhance our GDSS from text-based to multimedia-based.

Finally, it is needed to incorporate existing decision-related technologies into our system, for example, automatic idea categorization or intelligent voting method selection. Many research have been done on specific parts of GDSS and many technologies are present now.

【References】

[1] DeSanctis, G. and B. Gallupe., "Group Decision Support Systems: A New Frontier," Management Science, May 1987.

[2] Huber, G.P., "Issues in the Design of Group Decision Support Systems," MIS Quarterly, September 1984.

[3] Buckley, S. R. and Yen, D., "Group Decision Support Systems: Concerns for Success," The Information Society, Vol. 7, pp. 109-123, 1990.

[4] Vogel, D., et al., "Group Decision Support Systems: Determinants of Success," Transactions DSS 87, San Francisco, June 1987.

[5] Graham, I., Object Oriented Methods, 2nd Edition, Addison Wesley, 1993.

[6] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W., Object-Oriented Modeling and Design, Prentice Hall, 1991.

[7] Park, Olfman, and Satzinger, "Attitude Toward and Preference of Group Work," working paper in Claremont Graduate School, 1995.

[8] Turban, E., Decision Support and Expert Systems: Management Support Systems, 3rd edition, Macmillan Publishing co., p.355, 1993.

[9] Mandviwalla, M. and Olfman, L., "A Synthesis of Generic Groupware Design Requirements from a Multi-disciplinary Survey and Analysis of Work Group Research," Programs in Information Science, The Claremont Graduate School, 1994.

[10] DeSanctis, G. and Gallupe, B., "A Foundation for the Study of Group Decision Support Systems," Management Science, Vol. 33, No. 5, pp.589-609, 1987.

[11] Nunamaker, J. F., Dennis, A. R., Valacich, J. S., Vogel, D. R., and George, J. F., Group Support Systems Research: Experience from the Lab and Field. In Jessup, L. M., Valacich, J. S., Group Support Systems: New Perspective, Macmillan Publishing co., 1992.

[12] "GroupSystems V: Basic Tools Manual," Ventana Corporation, 1992.

[13] Perin, C., "Electronic Social Fields in Beuraucracies," Communications of the ACM, Vol. 34, No. 12, pp.75-82, 1991.

[14] Kling, R., "Cooperation, Coordination, & Control in Computer Supported Cooperative Work," Communications of ACM, Vol. 34, No. 12, pp.83-88, 1991.

[15] Steiner, I. D., Group Process and Productivity, New York: Academic Press.

[16] Hackman, J. R., The Design of Work Teams. In J. Lorsh (Ed.) Handbook of organizational behavior, pp.315-342, Englewood Cliffs, NJ:Prentice Hall, 1987.

[17] McGrath, J. E., "Time matters in groups," In J. Galegher, R. Kraut, & C. Egido (Eds.), Intellectual Teamwork:Social and Technological Foundations of Cooperative Work, pp. 23-78, Hillsdale, NJ: Lawrence Erlaum, 1990.

[18] Gersik, C. J. G., "Time and Transition in Work teams: Toward a New Model of Group Development," Academy of Management Journal, Vol. 31, pp.9-41, 1988.

[19] Gersik, C. J. G., "Marking Time: Predictable Transitions in Task Group," Academy of Management Journal, Vol. 32, pp.274-309, 1989.

[20] Tuckman, B. W., "Developmental Sequence in Small Groups," Psychological Bulletin, Vol. 63, pp. 384-399, 1965.

[21] Rumbaugh, J., "OMT: The Development Process," Journal of Object-Oriented Programming, 1995.

[22] Yourdon, E., Modern Structured Analysis, Yourdon Press, 1989.

[23] DeMarco, T., Structured Analysis and System Specification, Prentice Hall, Englewood Cliffs, NJ, 1979.

[24] Arnold, Bodoff, S., Coleman, D., Gilchrist, H., and Hayes, F., "An Evaluation of Five Object-Oriented Development Methods," Journal of Object-Oriented Programming, Special Issue on Analysis & Design, pp.107-121, 1991.

[25] Wirfs-Brock, R., Wilkerson, B., and Wiener, L., Designing Object-Oriented Software, Prentice Hall, Englewood Cliffs, NJ, 1990.

[26] Karam, G. and Casselman, R., "A Cataloging Framework for Software Development Methods," IEEE, February 1993.

[27] Buhr, R., Practical Visual Techniques in System Design: with Application to Ada, Prentice Hall, Englewood Cliffs, NJ, 1991.

[28] Booch, G., Object-Oriented Analysis and Design with Applications, Second Edition, Benjamin/Cummings, 1991.

[29] Silver, M. S., "Decision Support Systems: Directed and Non-Directed Change," Information Systems Research, Vol. 1, No. 1, pp. 47-70, 1990.

[30] Jablin, F. M. and Seibold, D. R., "Implication for Problem Solving Groups of Empirical Research on 'Brainstorming': A critical review of the literature," The Southern States Speech Communication Journal, Vol. 43, pp.327-356, 1978.

[31] Nunamaker, J. F., Dennis, A. R., Valacich, J. S., Vogel, D. R., and George, J. F., "Electronic Meeting System to Support Group Work," Communications of ACM, Vol. 34, No. 7, pp.40-61, July 1991.

김성회

1973년 서울대학교 공과대학 학사
1978년 미국 Univ.of Missouri-Columbia 산업공학 석사
1983년 미국 Stanford 대학원 경영정보학 박사
현 재 한국과학기술원 테크노경영대학원 경영정보전공 교수, 대한의료정보학회 이사, 한국 CALS/EC 협회, 기술협회 이사, 한국 CALS/EC 협회, 기술협회 전문가위원회 부의장, 국가정보화 추진위원회 자문위원
관심분야 CALS, GDSS, BPR(Business Process Reengineering), Decision Analysis, Supply Chain Management

조성식

1994년 고려대학교 전산과학과
1997년 한국과학기술원 경영공학 석사
현 재 한국과학기술원 경영공학 박사과정
관심분야 Group Decision Support System, Supply Chain Management, CALS

김선욱

1979년 고려대 공대 산업공학과 학사
1981년 고려대 대학원 산업공학과 석사
1990년 미국 Oregon 주립대 산업공학과 박사
현 재 단국대 산업공학과 부교수
관심분야 정보시스템, 인공지능, 전문가시스템, 생산관리 등

박홍국

서울대학교 경영대학 졸업, 같은 학교 대학원 수료. 미 클레어몬트 대학에서 경영학 석사 및 경영정보학 박사학위를 취득하였다. 미국에 건너가기 전, 대통령 비서실에 근무하기도 했으며 귀국 후에는 대우통신, 대우조선 MIS실 등 기업체의 실무 책임자로 있으면서 경영혁신 프로젝트를 수행하였고 현재 상명대학 정보통신학과 교수이며, 전산정보대학원장으로 재직중이다. 저서로는 「정보기술과 의사결정」, 「중역정보시스템」, 「의사결정지원시스템」이 있다.

96년 9월 최초 접수, 96년 10월 최종 수정